A Multi-level Optimization Strategy to Improve the Performance of Stencil Computation

Gauthier Sornet, Fabrice Dupros, and Sylvain Jubertie

Univ. Orléans, INSA Centre Val de Loire,
LIFO and BRGM.

Motivations

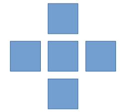
- Many applications widely use stencil computation :
  - Image : Convolution filters
  - PDE solvers : Fullswof, EFISPEC 3D,...
- Stencil computation represents a major part of the total computing elapsed time.
- Stencil computation has a low arithmetical intensity.
- Compilers optimisations could be limited : unrolling, vectorization...

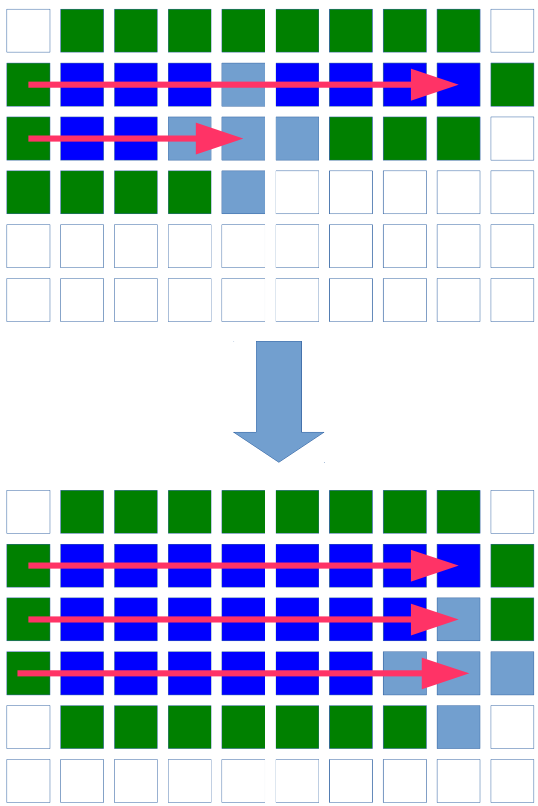
Outline

7pts and 27pts Stencils

2D stencil

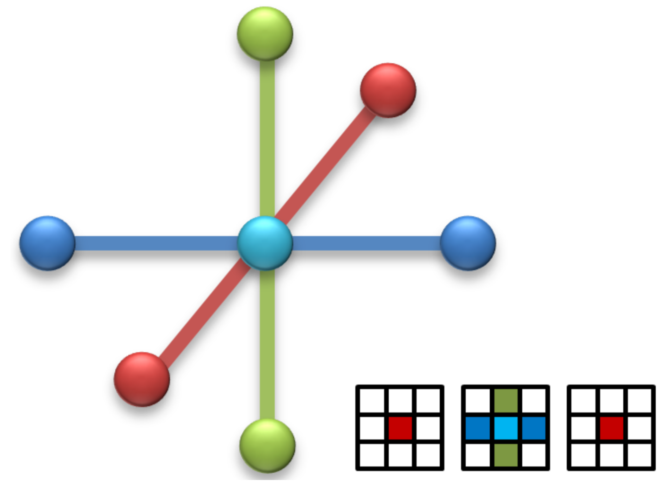
5pts stencil



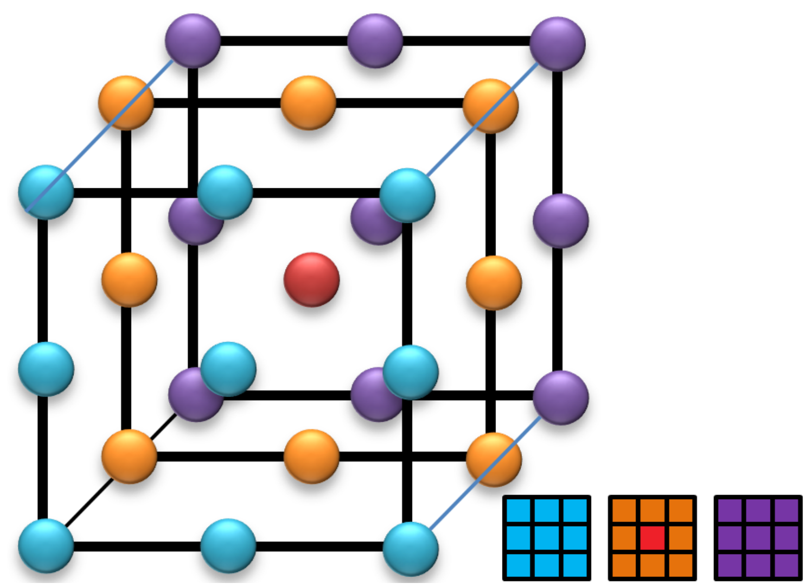


3D stencil

7pts stencil



27pts stencil



Reuse intensity

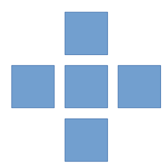
Caracterization

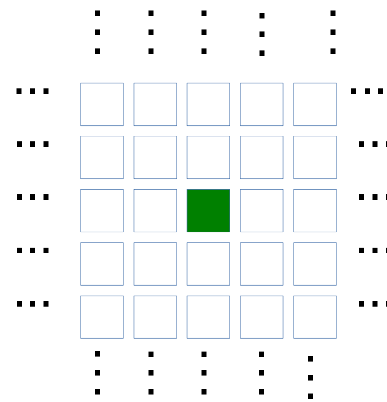
Stencil has a low compute / memory ratio.
We need a metric to caracterize stencil types.

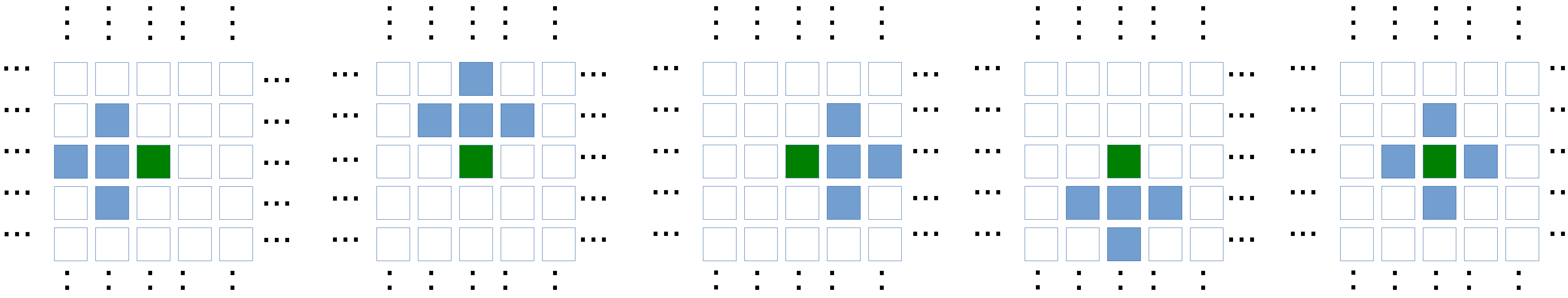
Definition

The reuse intensity is the factor between memory reuse times and the number of loaded bytes for each iteration.

Exemple

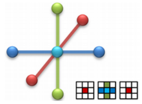
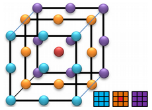






Reuse intensity

R.I of the stencil under study

Representation	Name	Number of cells	Number of operations by cell	R.I
	7pts	7	1	7*1/4=1.75
	27pts	27	1	27*1/4=6.75

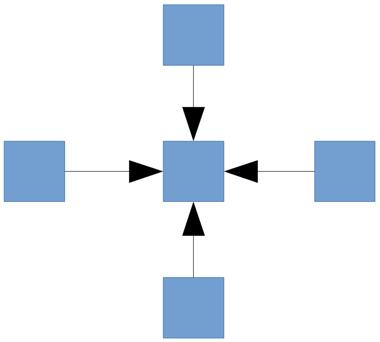
Manual Vectorization

Compilers do not garantee vectorization.
We explicitly use SIMD instructions to force the compiler to use it and to optimise with it in our way.

Our computing plaforms are able to compute 8 floats of 32bits by AVX instruction.

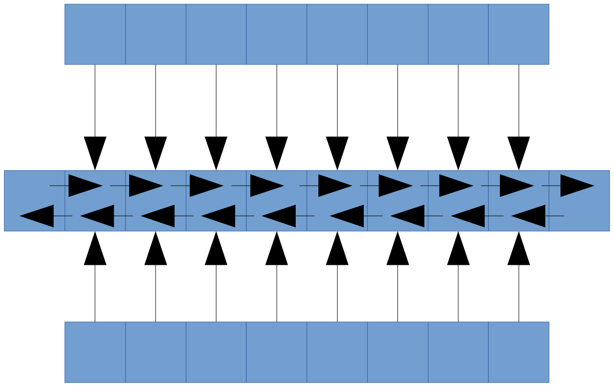
SISD add (not vectorized instructions) :

5 cells added at a time.



SIMD add (vectorized instructions) :

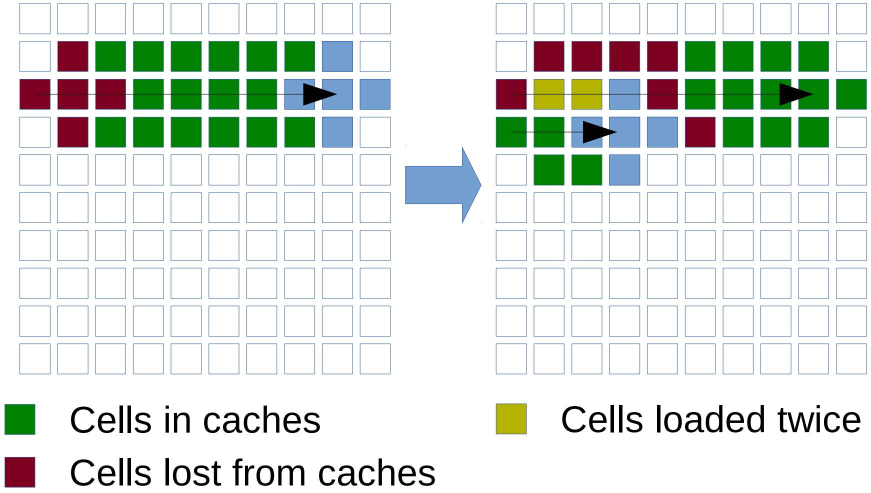
8*5 cells added at a time.
Theoretical speed up of 8.



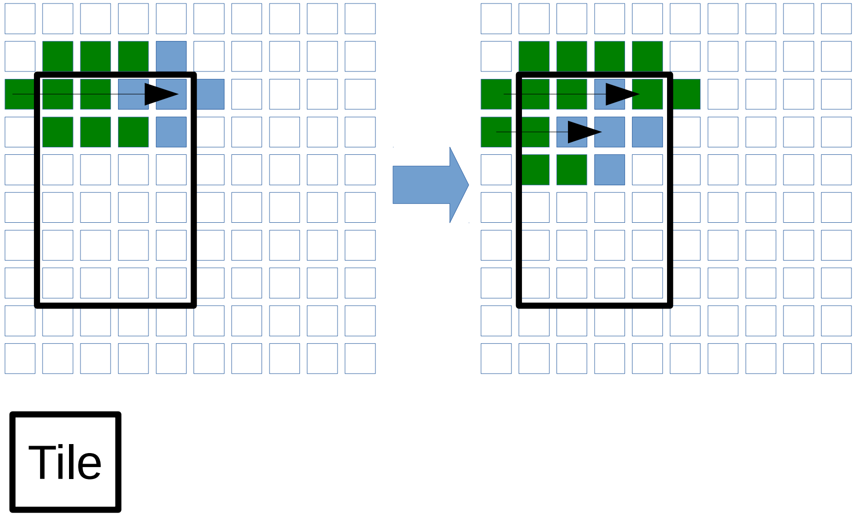
Tiling

- Crossing domain with tiling
- Space tiling
- Time tiling
- Memory, Cache optimisations

no Tiling

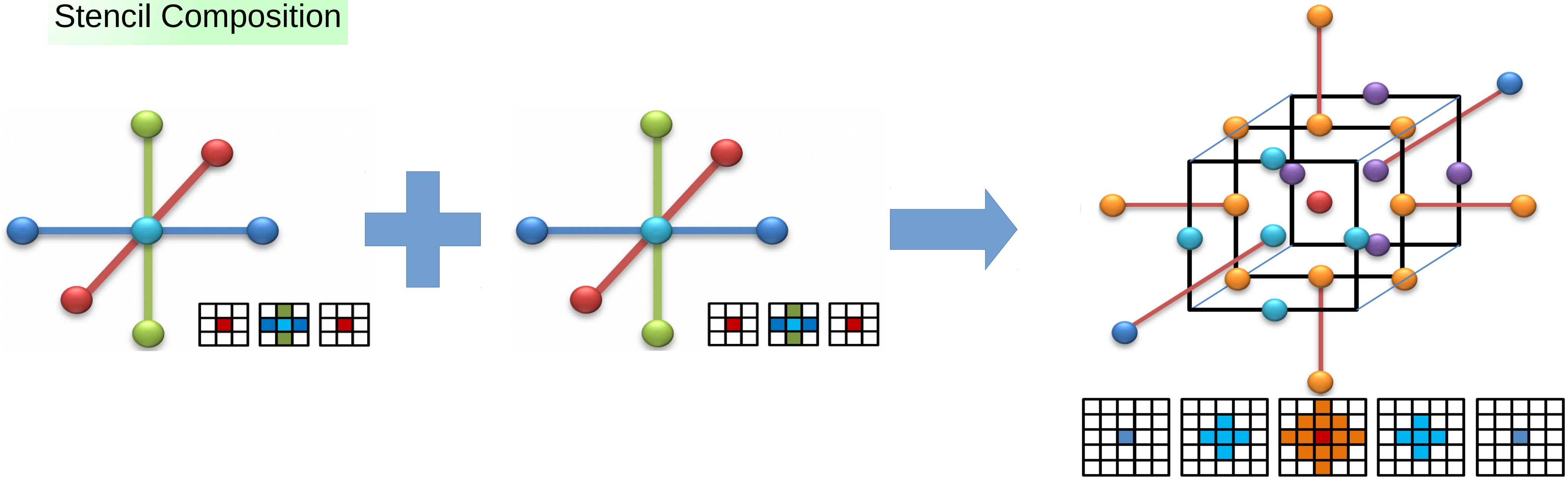


Space Tiling

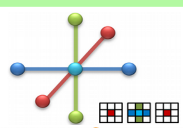
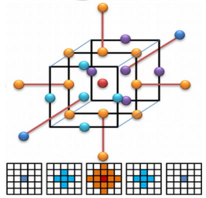


Optimizations

Stencil Composition



R.I of the stencil under study

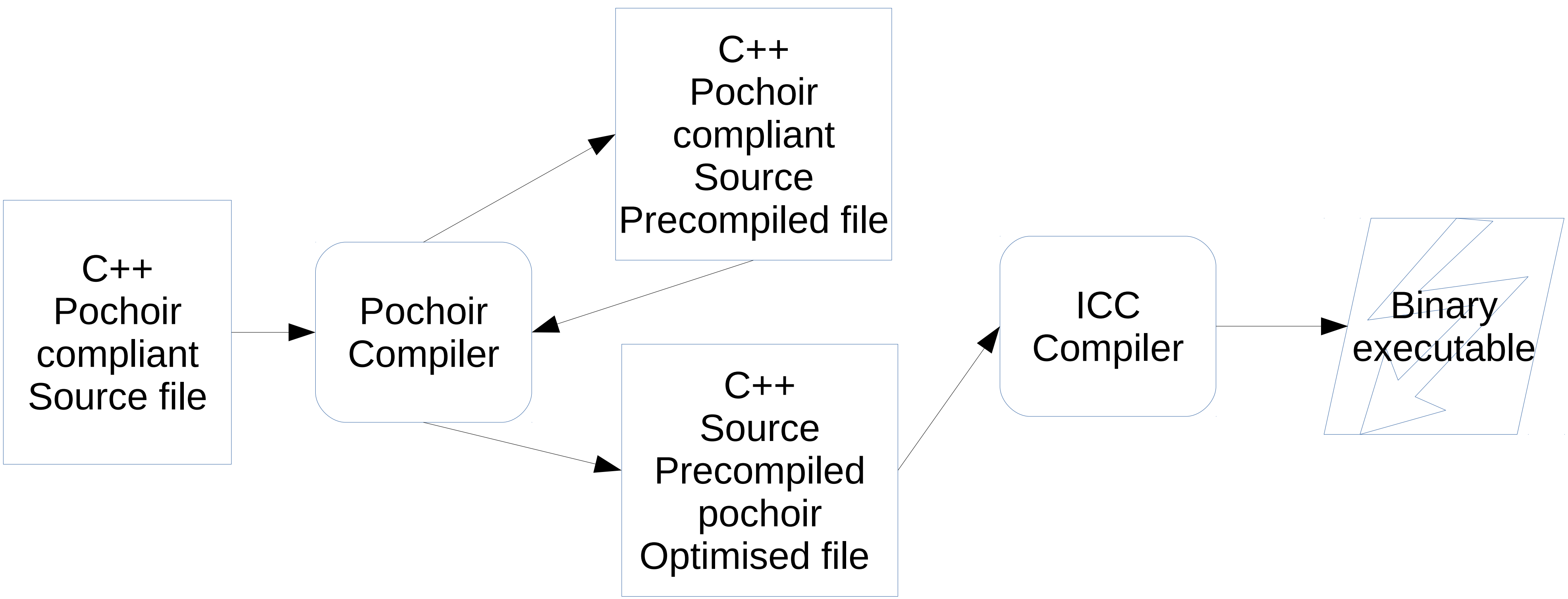
Representation	Name	Number of cells	Number of operations by cell	R.I
	7pts	7	1	7*1/4=1.75
	25pts	25	1	25*1/4=6.25

The Pochoir library

References

Yuan Tang, Rezaul Alam Chowdhury, Bradley C. Kuszmaul, Chi-Keung Luk, and Charles E. Leiserson.
The pochoir stencil compiler. SPAA ’11.

DSL base

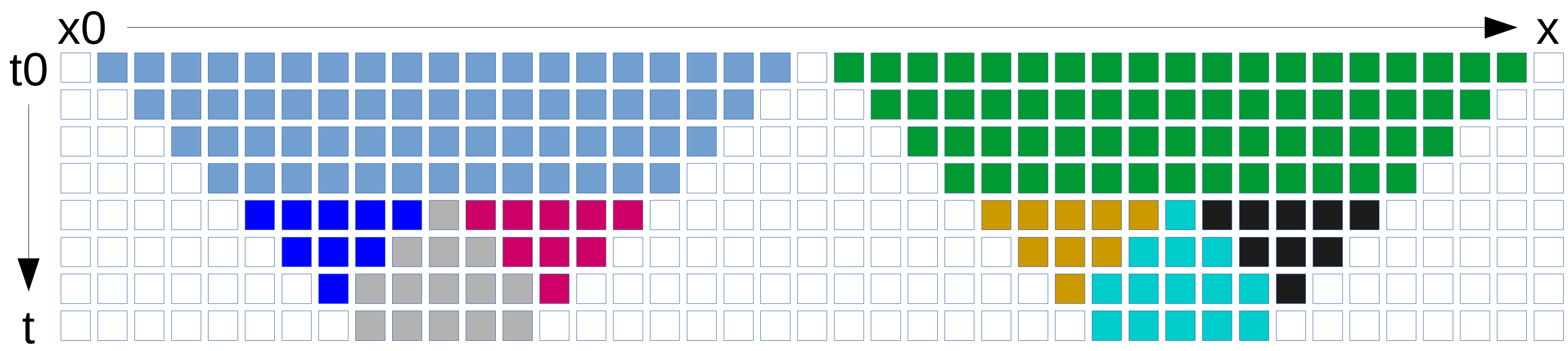


Hyperspace cut algorithm

Two optimization levels :

- Independant Cilk tasks cutting to reduce thread synchronisation
- Time and space cutting to optimise caches

Let consider a one x dimension domain to compute t times :



- What about vectorization ?

Experimental results

Architectures

- Intel Xeon E5-2697 v2 Ivybridge processors, for a total of twenty-four cores at 2.7Ghz.
- Intel Xeon E5-2697 v4 Broadwell processors, for a total of eighteen cores at 2.3Ghz.

Compilers

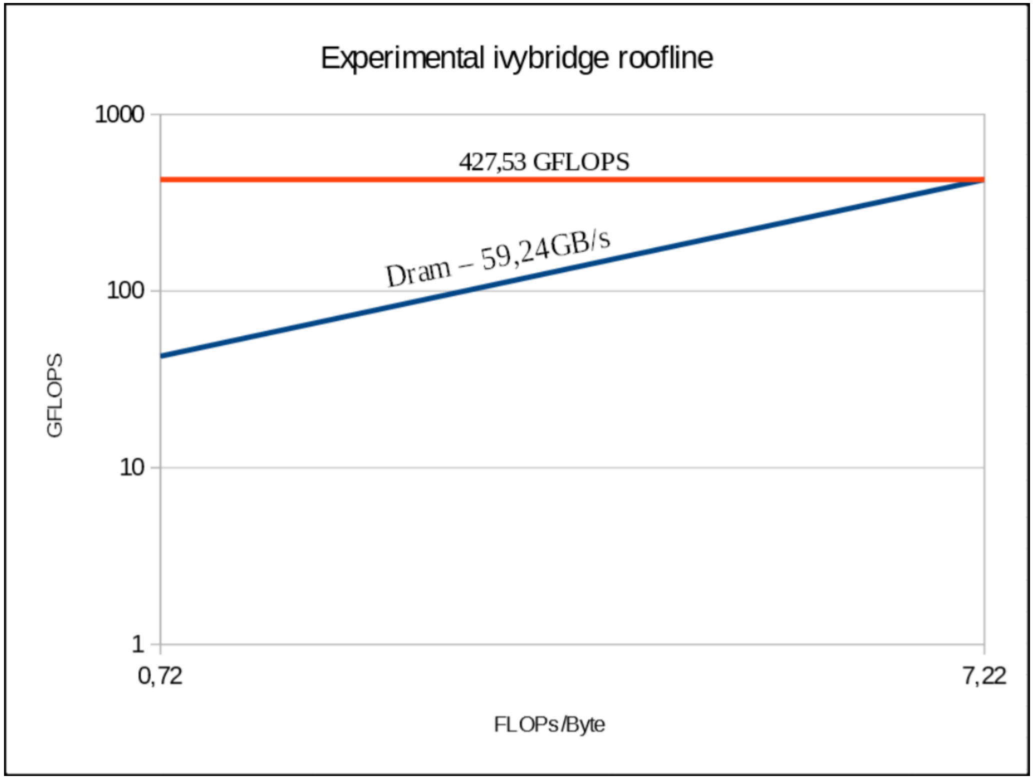
Clang 3.8, GCC 6.2 and ICC 17 with -O3 -march=native optimization flags and OpenMP multi-threading.

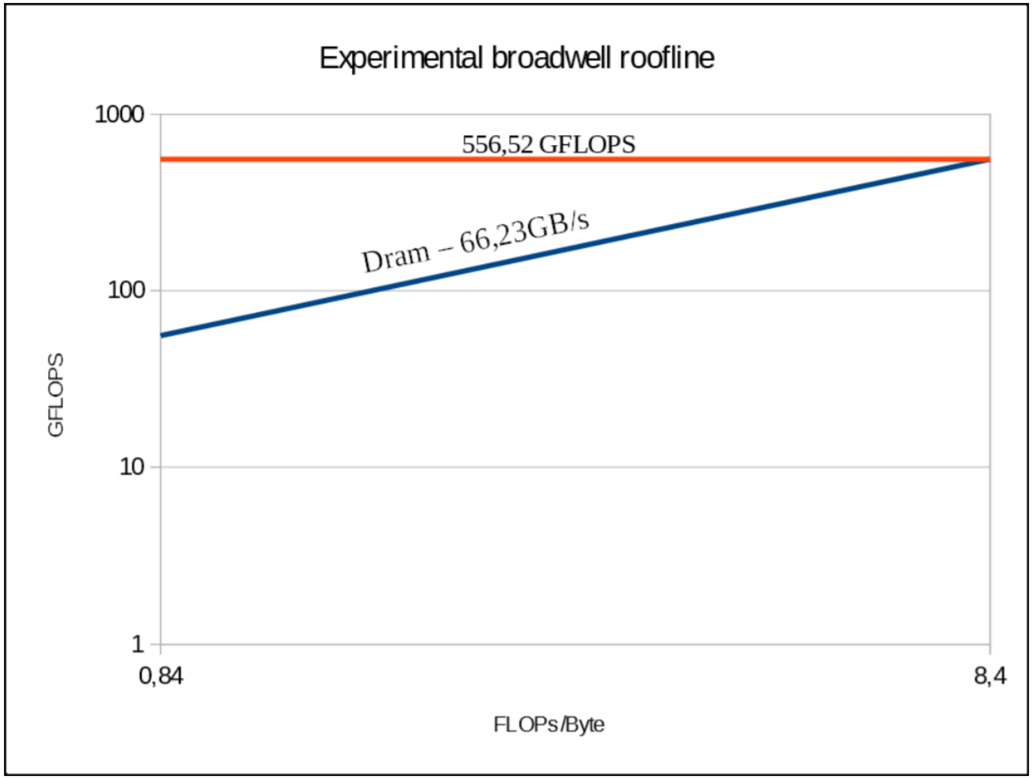
Fixed parameters

- Single float precision
- 512x512x512 grid
- 100 iterations
- Each results from 10 runs

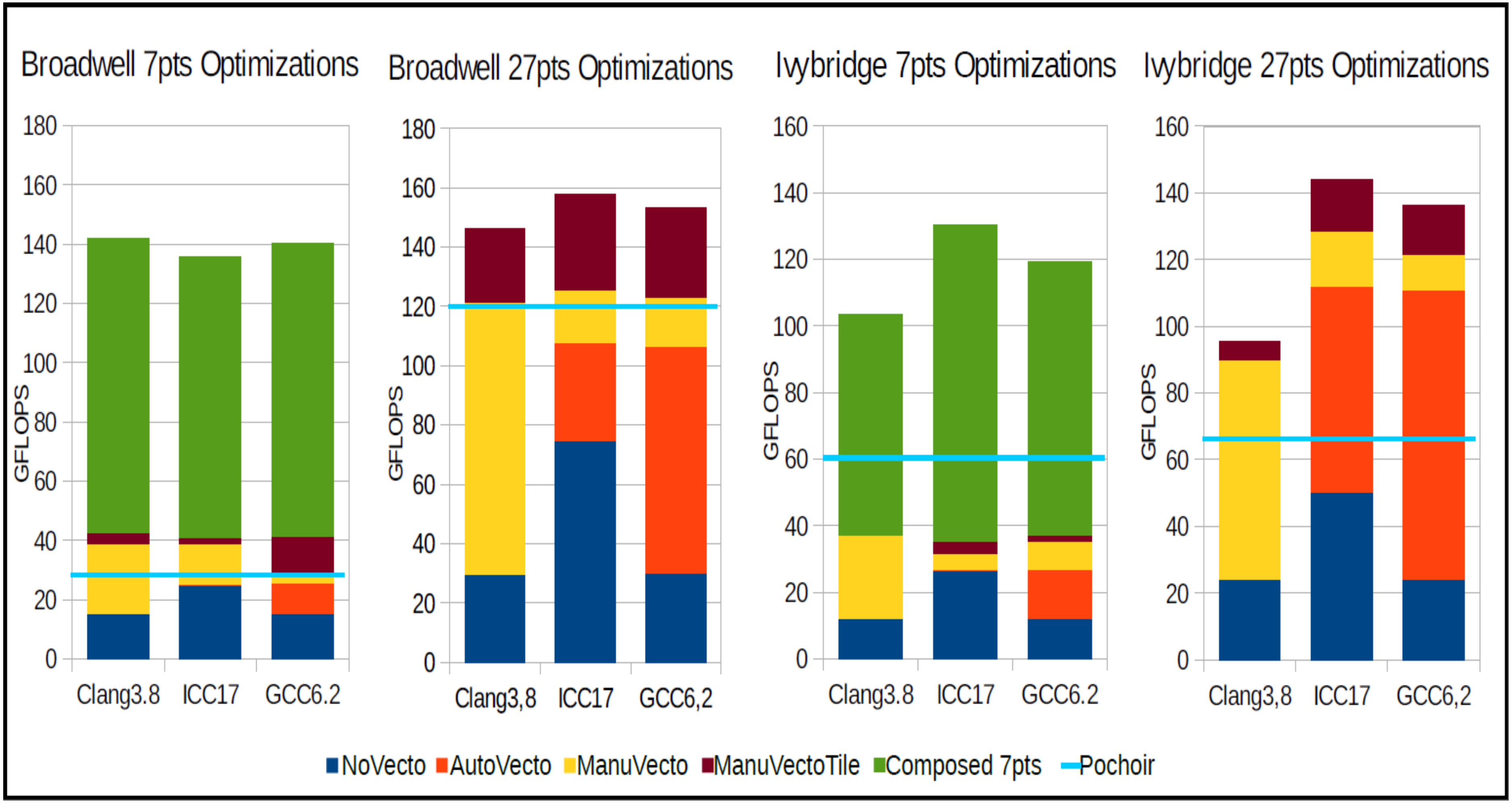
Experimental results

Rooflines



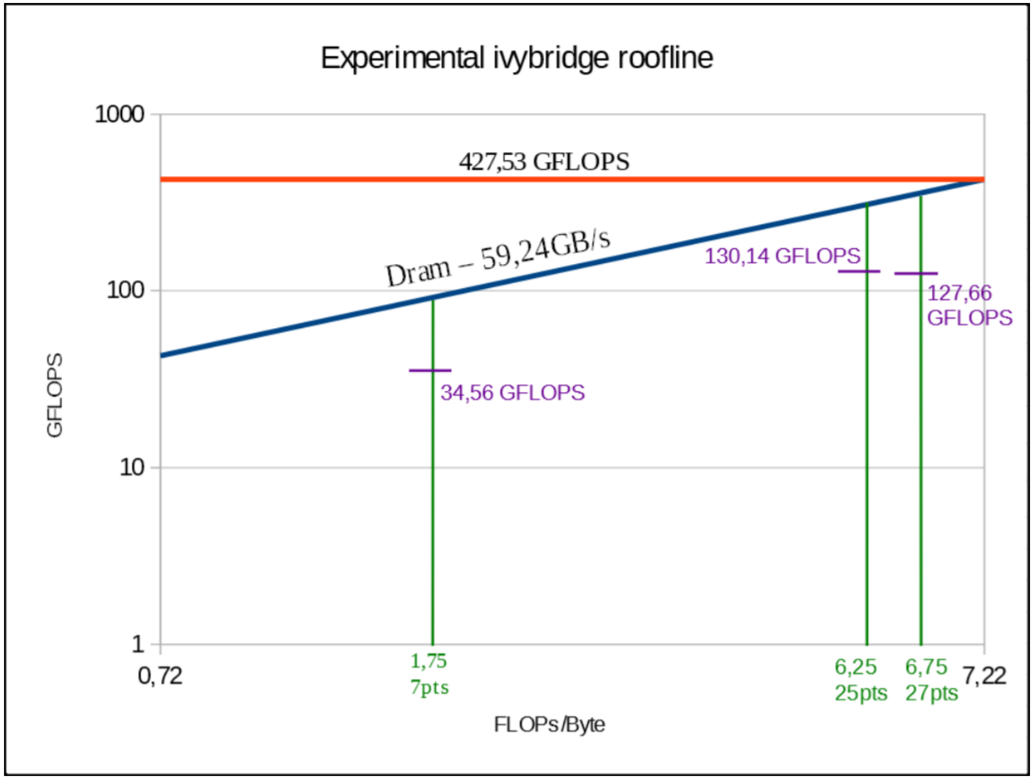


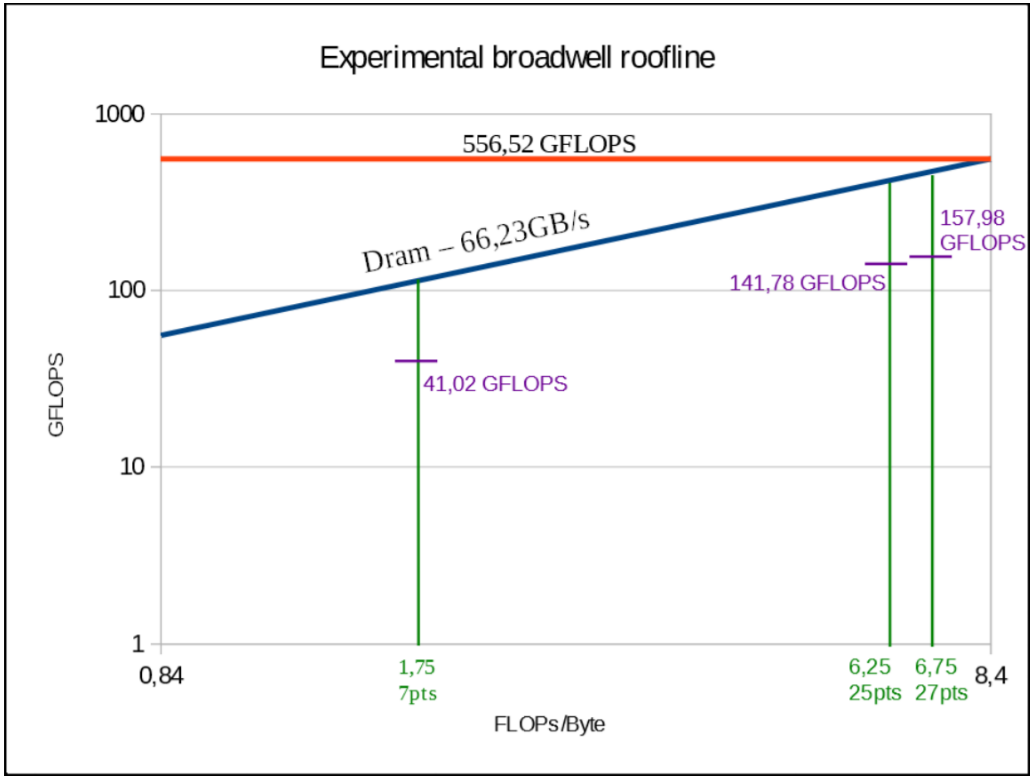
Performances



Experimental results

Rooflines





Experimental results

Elapsed time

Broadwell

Compilateur	7pts 100it	25pts 50it	Poch 7pts 100it	Poch 25pts 50it
Clang3.8	1875ms	937,5ms		
ICC17	1956ms	1160ms	2809,55ms	2300,745ms
GCC6.2	2117ms	1276ms		

Ivybridge

Compilateur	7pts 100it	25pts 50it	Poch 7pts 100it	Poch 25pts 50it
Clang3.8	2157ms	1519,5ms		
ICC17	2288ms	1305,5ms	1315,62ms	2534,22ms
GCC6.2	2174ms	1338,5ms		

Conclusion

- Simple optimizations but hard to make them work together.
- Experimental results follow the R.I analyses.
- Pochoir relies on ICC vectorization and optimization.
- The stencil composition is interesting.

Perspectives

- Automatic optimization.
- DSL or DSEL.
- Specific compiler or meta-programming.

Any question ?